Appendix A

Class AID

```
java.lang.Object
  |
  +--javacard.framework.AID

|
        +--com.sun.smartcard.AID
```

All Implemented Interfaces:

Readable, java.io.Serializable

Direct Known Subclasses:

SunAIDEncoding

---

public class **AID**

extends javacard.framework.AID

implements Readable, java.io.Serializable

A representation of the Application Identifier with support for a Java Card standardized encoding of additional data in the **PIX** field. This class and the complementary `AID interpreter` class provide a key functionality for generic support of card application management and application interoperability in the Java Card application framework. This class explicitly provides enhanced support for the selection of `Java Card Applets` by the terminal using a partial AID.

In particular this class provides support for the following distinct data elements associated with the Applet which are encoded in the ISO standardized **AID** `byte[]` representation (see below for the default encoding scheme):

1. It specifies the application provider as a legal entity (via the **RID**), as defined by ISO/IEC 7816-5. By convention, in the Java Card implementations only nationally or internationally registered **RID**s are formally supported.

2. It specifies the in-card instance of the Java Card *"firewall"* that contains the Applet and all the objects the Applet uses to store its data. Or, in more appropriate terminology, it specifies the *"owner"* of the Applet and Applet data objects instances in the card. The owner of an applet is commonly referred to as the Application Provider. To complicate matters, in principle, the legal

entity identified by the **RID** in an **AID** may be present in the card as multiple different *owners*. For example the RID may identify an organization that is both the card issuer and provider of an application. In this example the card management operations performed by the card issuer applet should be separated by a firewall from the application operations implemented by the application Applet. The Ownership information is encoded in the **PIX** field as an extension of the **RID**.

3. It specifies a unique applet within a firewall, distinguishing it from any other applets that may be instantiated in the card with the same owner. This information is used for selection of applets. This information is encoded in the **PIX** field as an extension of the owner.

4. It specifies specific functions for applets of the same *owner*. For instance with distinct functions possibly indicated for a pre-paid (purse), debit (ATM) card or credit card applet. Similar function indications have been adopted by ISO 14443 for contact less cards as Application Function Identifiers (AFI). The interpretation of this code can be standardized by industry segment bodies. This information is contained in the **PIX** field.

5. It may specify additional information on the applet code, like code version numbers, for instance to complement the specified function. This information is optional. This optional information will be encoded in the **PIX** field.

6. It may specify additional information on the applet data instantiated in the card like key (batch) identifiers, or configuration data like the number and kind of supported data sets. This **AID** data element refers to Applet instance data that may vary per applet instance, but that does not, or rarely change over the life time of that instance. This data may be used by the applet at installation time to configure it self. When retrieving an AID from the card part of the code space for this data may be used to encode applet state data (see next item) This optional information will be encoded in the **PIX** field.

7. It may specify additional information on the applet instance state data like the currency and value of the balance in a purse, or a currently assigned "service level" as may be found in a loyalty scheme. This **AID** data element refers to Applet instance data that is different in each applet instance and that may vary in each selection of the applet. This optional information will be encoded in the **PIX** field. The encoding space for this data in the PIX may be the same (as a part of) the configuration data (see previous item).

In this design each applet provides its own derived class with appropriate implementations of the abstract methods. The OwnerID is different from the RID as according to ISO rules an organization can only obtain a single RID. But, the card issuer may also be the provider of an applet on the card but may want to specify different owners for the loading Applet (function) and the other applet. And also, the issuer may want to distinguish ownership of library packages from both the loading applet and the other applet. Similarly, the

AppletID is different from the OwnerID, as the same owner may have more than one collaborating applet, where each applet is specified by the owner as selectable separately.

Encoding and Interpretation

The encoding and interpretation of the data elements actually contained in an **AID** may in general be specific to the application provider (registered entity). However, this `AID` class provides a default encoding for the first, non optional, data elements contained in the **PIX** as described below. The encoding of optional data elements in the **PIX** field is not defined, and must be specified in a sub class of this class when such optional data is required. This class provides the framework for such optional data to be used. The in-card interpretation of the optional instance data, and possible use of this data for install-time configuration is not specified here. However, an abstract class is provided as a basis for the representation in the card of such configuration information.

Interpreting the **AID** information in a terminal may be accomplished by using the `AID interpreter` instance. Typically, a sub class of the `AIDInterpreter` class will be used to represent any of the optional data elements. The AID interpreter may be provided to the terminal via a provisioning mechanism that uses the unique value of the RID in any given AID. Interpreting the data encoded in the **AID** by that class might involve retrieving data from the world wide web.

Applet Selection, Firewall and Ownership

The `AID` class provides explicit support for *selection of applets by partial **AID*** with the implementation of the methods `#matchId()`, `#matchOwner()` and `#matchRID()`. The mechanism provided by these methods addresses the issue of partial matching which has been left underspecified in version 2.1 of the Java Card technology specifications.

An implementation of the `Select Applet command` may use these methods by examining all Applets in a card in a sequence of increasingly less specific matches. First the applets are tested for a matching applet identifier, using the `matchId()`. If no Applet has been identified the applets are tested for a match on the owner ID using `matchOwner()`. And finally, if that does not identify a single Applet, all Applets are tested for a match on the **RID** using `matchRID()`. This three stage matching algorithm provides full functional backward compatibility with Applets that are implemented with the `javacard.framework.AID` class (version 2.1).

Encoding

The `AID` class provides a default encoding of the **AID** in a sequence of maximally 16 bytes. The encoding follows the following rules (see the diagram):

1. The ***RID*** is encoded as defined by the international standard ISO/IEC 7816-5 over the first five

bytes of the AID encoding. The `AID` implementation fully conforms to this international standard.

2. The *OwnerID* is encoded as the first six bytes of the AID encoding, thus including the **RID**. The value encoded in the additional byte is not specified here and will be allocated by the entity identified by the **RID**. The purpose of this value is to distinguish between the different commercial and/or functional roles the registered entity may have in the card that require separation by a firewall.

3. The *ApplicationID* is encoded as the first 7 bytes of the AID encoding, thus including the **OwnerID**. The value encoded in the additional byte is not specified here and will be allocated by the entity identified by the **RID**. The purpose of this value is to distinguish between specific applets on the card that share the same owner and firewall, such as a Debit Card applet and a Credit Card applet provided by the same financial institution.

4. The *Application Function ID* is encoded on the 8th byte in the AID encoding. The value of this byte is encoded according to international standard ISO/IEC 14443 (contact-less cards).

5. The optional applet version, configuration and applet state data are encoded over the remaining 8 bytes of the AID. The applet AID matching algorithm that may be used in applet selection excludes these data elements. There is no default encoding for these optional data elements.

    - An encoding for the configuration data is specified by the implementation of a sub class of the inner class `AID.ConfigurationData` The concrete configuration data class in the Applet may be used at applet-install time only or, in the implementation of a possible APDU handler to decode configuration data during the operational phase.

    - Using Applet state data in an AID retrieved from the card is supported with a `method call`. This method has a default empty implementation. The method is intended to append a `byte[]` representation of the selected Applet state data to the **AID** encoding when it is being read by a terminal.

    In order to use either version data, configuration data or applet state data a sub class of this class must be defined that overrides the relevant methods, `getVersion()`, `getConfiguration()` and `readStateBytes()`, respectively.

The default encoding of the data elements defined by the AID java class is summarized in the following table.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
RID					PIX										
OwnerID															
AppletID							AF	Optional Data Elements							

AF=Application Function ID

Java Card Specific Encoding

The specification of the `AID` class supports enhanced interoperability in application selection and card management with a set of reserved encoding constants and corresponding standard decoding methods. This allows card applications to identify specific common roles in the card by examining the AID of applications. In the default encoding of the OwnerId and the Application ID the values 0x00 and 0xFF have been specified for this purpose. The default encoding implemented in this class uses these constants, implementations of alternative encoding schemes may use different values. The meaning of the special values can be asserted with:

`OwnerId.#isIssuer()`

This applet is controlled by the card issuer. It may be the applet management Applet, or an other card management related applet. The default coding value is **0XFF**.

`OwnerId.#isNotControled()`

This applet is not actively controlled by its owner as may be specified by its AID. It may be an applet that contains personal data related to and entered by the cardholder. The default coding value is **0X00**.

`AppletId.#isManager()`

This Applet is the card's applet management Applet or a delegated management Applet. The Application Management Applet must be owned by the card issuer. The default coding value is **0XFF**.

`AppletId.#isInformation()`

This applet provides information, e.g. pertaining to the cardholder. In addition to returning `true` in this method a cardholder data applet may indicate itself as not being controlled. The default coding value is **0X00**.

Alternative encoding, e.g. to allow more space for optional data elements may be provided by overriding the relevant methods in this class. Any coding scheme should fully support the non optional data elements and their specific function codes.

Implementation Design

The `AID` class is implemented as wrapper around a data representation as a `byte[]`. Accessor methods and inner classes are specified to access and represent the types of the data elements that may be represented by an instance of the `AID` class. The implementation choice for the `byte[]` data representation is intended to improve the speed of retrieval of an AID. This retrieval may happen after the chip is reset when a list of all AIDs in the card may be assembled. Also, this data representation greatly simplifies the implementation of different encoding rules in possible subclasses.

The implementation of the classes to represent the component data elements similarly is as wrappers around `byte[]`. In the implementation instances of these classes share their data representation with their parent AID instance. With some increase in code complexity this design may be advantageous in speed and storage requirements. Security in the use of the AID is also enhanced, in that after the value of an `AID` instance has been committed to the non volatile card memory during Applet installation the information in it is never changed over the lifetime of an Applet. The implementation of `AID` and its component data type classes as a wrapper leverages the functionality provided by `Buffer` class that implements a generic wrapper around a `byte[]`. The `Buffer` is actually used as the base class for all these classes and supports the access to a shared `byte[]`. The shared `byte[]` holds the single constant data representation of the `AID` instance in the card memory. These data component types completely encapsulate the actual encoding and expose only the required semantic functionality.

The types defined for the data elements are:

1. `AID.RID`,
2. `AID.OwnerId`
3. `AID.AppletId`

4. AID.ConfigurationData

The classes for the AID component data elements also implement the Readable interface (via the base class ReadableBuffer) to facilitate retrieval as ISO files.

The AID class is an extension of the class and provides source code compatibility with existing applet implementations. Functionally, however, this class supersedes its base class completely.

Java Card Converter issues

The AID class implements the Serializable interface to support full card emulation with persistent in-card state. The implementation also provides methods to parse and produce a human readable form of the AID. These methods utilize in their implementation classes that are not part of the JCRE (StringTokenizer). With these features the implementation assumes the Java Card convertor to silently remove the interface and the non-convertible methods from the converted in-card class definition.

Note

This class, com.sun.smartcard.AID, extends the javacard.framework.AID class only to provide backward compatibility with existing card platform implementations, functionally this class replaces the original base class completely.

Nested Class Summary	
Static class	**AID.AppletId** Represent the unique reference to an Applet that is contained in an AID.
Static class	**AID.ConfigurationData** Represent the Applet instance configuration data that may be contained in an AID.
Static class	**AID.OwnerId** Represent the owner of an Applet instance that is contained in an AID.
Static class	**AID.RID** Represent the RID (Registered ID) part of an AID.

Nested classes inherited from class com.sun.smartcard.util.Readable
Readable.WithStream

Field Summary	
private byte[]	**data** The data representation of an AID is a byte[].
private static short	**SIZE** The size of an AID is fixed.

Constructor Summary	
protected	**AID**() Create an Empty AID.
	AID(AID.RID rid) Create an AID with a specific RID encoding.
	AID(byte[] encoding) Create an AID from a specific full length encoding.

Method Summary	
(package private) AID.AppletId	**getAppletId**() Get the AppletId part from an AID.
Protected AID.RID	**getAssignedRID**() Obtain the RID assigned to a card issuer or an application provider.
AID.ConfigurationData	**getConfiguration**() Obtain the configuration data part from the AID.
byte	**getFunctionCode**() Obtain a descriptor of the function of the applet conforming to ISO/IEC 14443 This method uses the default encoding, with the one byte code immediately following the AID.AppletId.
(package private) AID.OwnerId	**getOwnerId**() Get the OwnerId part from an AID.
(package private) AID.RID	**getRID**() Get the RID part from an AID.
short	**getVersion**() Obtain a descriptor value indicative of the version of the applet code.
boolean	**matchId**(byte[] encoding, short offset) Compare the AID with a given encoding as used when the Applet

	instance that holds the AID is considered for "selection by AID".
boolean	**matchOwner**(byte[] encoding, short offset) Compare the AID with a given encoding as used when the Applet instance that holds the AID is considered for "selection by AID".
boolean	**matchRID**(byte[] encoding, short offset) Compare the AID with a given encoding as used when the Applet instance that holds the AID is considered for "selection by AID".
static AID	**parse**(java.lang.String data) Interpret a comma separated list of numbers in a string as an AID.
short	**read**(byte[] buffer, short offset, short dataskip, short length) Append a serialized representation of the AID to a given byte[].
protected short	**readStateBytes**(byte[] result, short off, short length) Append the byte[] representation of the current Applet State to the AID's linear representation.
protected void	**setdata**(byte[] data) Set the data representation for this AID to the specified array of bytes.
java.lang.String	**toString**() Represent the AID as a string of up to 16 hexadecimal integers.

Methods inherited from class javacard.framework.AID
Equals, equals, getBytes, partialEquals, RIDEquals

Methods inherited from class java.lang.Object
Clone, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

SIZE

```
private static final
short SIZE
```

The size of an AID is fixed.

data

```
private byte[]
data
```

The data representation of an AID is a `byte[]`. This representation storage is shared by all the data elements that are encoded in it.

Constructor Detail

AID

```
protected
AID()
```

Create an Empty AID. This constructor is for use in subclasses; subclasses constructed with this constructor need to use the `set data()` to initialize it properly. This definition allows the use of anonymous inner classes in the implementation of a specific Applet subclass to implement an AID. In this way the implementation of the `readStateBytes()` method can be done with direct access to relevant state objects.

AID

```
public AID(AID.RID rid)
```

Create an AID with a specific RID encoding.

AID

```
public
AID(byte[] encoding)
```

Create an AID from a specific full length encoding. An encoding longer than 16 bytes is silently truncated, an encoding shorter than 5 bytes is an error.

Method Detail

parse

```
public static AID
parse(java.lang.String data)
```

Interpret a comma separated list of numbers in a string as an AID.

Note

This method uses classes that are not part of the JCRE and as a consequence will be skipped by the Java Card converter when building the in-card definition of the AID class. As a consequence none of the in-card methods can refer to this method.

---

toString

```
public
java.lang.String toString()
```

Represent the AID as a string of up to 16 hexadecimal integers.

Overrides:

toString in class java.lang.Object

---

setdata

```
protected
final void setdata(byte[] data)
```

Set the data representation for this AID to the specified array of bytes.

getRID

```
final AID.RID
getRID()
```

Get the RID part from an AID.

Returns:

an instance of AID.RID that is wrapped around the shared data representation.

---

getAssignedRID

```
protected AID.RID
getAssignedRID()
```

Obtain the RID assigned to a card issuer or an application provider. This method is intended for use in the constructor of a subclass of the AID to assure consistency of the data received with the

representation specific to the sub class. To be used as intended this method should return the constant instance of a {link RID} class that is initialized with the **RID** prefix of **AIDs** that are decoded by the subclass.

Returns:

null, the base class does not check for its encoding as compatible with a specific RID.

getOwnerId

```
final AID.OwnerId
getOwnerId()
```

Get the OwnerId part from an AID.

Returns:

an instance of `com.sun.smartcard.AID.OwnerID` that is wrapped around the shared data representation.

---

getAppletId

```
final AID.AppletId
getAppletId()
```

Get the AppletId part from an AID.

Returns:

an instance of `AID.AppletId` that is wrapped around the shared data representation.

getFunctionCode

```
public byte
getFunctionCode()
```

Obtain a descriptor of the function of the applet conforming to ISO/IEC 14443 This method uses the default encoding, with the one byte code immediately following the AID.AppletId.

Returns:

a single byte containing the applet's function code.

getVersion

```
public short
getVersion()
```

Obtain a descriptor value indicative of the version of the applet code. The default encoding does not include a version number; by default this method returns 0.

Returns:

0, the default encoding implemented in the base class does not support a version.

getConfiguration

```
public AID.ConfigurationData
getConfiguration()
```

Obtain the configuration data part from the AID. During applet instantiation this data may be used to initialize the applet instance configuration. In addition, this object may include configuration data relevant to instance specific security initialization.

readStateBytes

```
protected short
readStateBytes(byte[] result,

short off,

short length)
```

Append the byte[] representation of the current Applet State to the AID's linear representation.

The default implementation does nothing. Applets that want to transfer dynamic state information in the AID can do so by overriding this method with an implementation that calls the appropriate (public, package) methods in the Applet subclass.

The method signature is compatible with File.Readable.read().

---

read

```
public short
read(byte[] buffer,
                  short offset,

short dataskip,

short length)
```

Append a serialized representation of the AID to a given byte[]. This method may be used to use a READ BINARY COMMAND to access the value of the AID.

This implementation is based on the default JavaCard **AID** encoding rules and its result is obtained by appending the results of the inherited read() for AppletId getVersion() and getConfiguration(), respectively, to the specified byte[]. When a subclass is used to implement an alternative encoding this method may need to be overridden.

The signature of this method is specified by the core version of the Readable interface method and its implementation utilizes the similar lesser OO interface for its AppletId and Configuration components. A more sophisticated design could utilize the Readable.WithStream.read(com.sun.smartcard.util.Stream.Writer) method which has a more OO signature.

WARNING:
This implementation is incomplete as it does not provide error and parameter checking; it is provided as example, e.g. of the use of the Readable interface.

Specified by:
read in interface Readable

Parameters:

dataskip - ***Ignored***

matchId

```
public final boolean
matchId(byte[] encoding,

short offset)
```

Compare the AID with a given encoding as used when the Applet instance that holds the AID is considered for "selection by AID". The comparison is restricted to the data included in the AppletID. If no applet is found with a match using this method, the list of applets will be searched again for a match using the matchOwner(byte[], short) method.

Note:
While the default encoding as provided by this class performs the comparison on a prefix of the byte[], alternate encoding schemes, may be implemented by a sub class which could involve possible discontinuous segments of the encoding. When such alternative encoding is used the partial matching may actually require a full, 16 byte data representation to be presented as the argument to this method. In the default encoding only a prefix that contains the applet ID is needed.

The actual matching is performed by the Buffer.match(byte[], short) method.

The input parameters include an offset to facilitate matching of the AID against the raw content of the APDU buffer.

Parameters:

encoding - a byte[] containing a possible encoding of an AID.

offset - the offset in the specified byte[] where the actual AID encoding starts in the encoding

Returns:

true iff the appletID part of this AID matches the specified encoding.

matchOwner

```
public final boolean
matchOwner(byte[] encoding,

short offset)
```

Compare the AID with a given encoding as used when the Applet instance that holds the AID is considered for "selection by AID". The comparison is restricted to the data included in the OwnerID; this method is typically called when no Applet instance is found that matches on the AppletID. If no applet is found with a match using this method, the list of applets will be searched again for a match using the matchRID(byte[], short)

Note:
While the default encoding as provided by this class performs the comparison on a prefix of the byte[], alternate encoding schemes, may be implemented by a sub class which could involve possible discontinuous segments of the encoding. When such alternative encoding is used the partial matching may actually require a full, 16 byte data representation to be presented as the argument to this method.

The actual matching is performed by the Buffer.match(byte[], short) method.

The input parameters include an offset to facilitate matching of the AID against the raw content of the APDU buffer.

Parameters:

encoding - a byte[] containing a possible encoding of an AID.

offset - the offset in the specified byte[] where the actual AID encoding starts in the encoding

Returns:

true iff the OwnerID part of this AID part matches the specified encoding.

matchRID

```
public final boolean
matchRID(byte[] encoding,

short offset)
```

Compare the AID with a given encoding as used when the Applet instance that holds the AID is considered for "selection by AID". The comparison is restricted to the data included in the **RID** part; this method is typically called when no Applet is found that matches on the Owner ID. If no Applet is found with a match using this method, then no applets in the card exist that match the given ID and the select by AID command will return a failure.

The actual matching is performed by the Buffer.match(byte[], short) method.

The input parameters include an offset to facilitate matching of the `AID` against the raw content of the APDU buffer.

Note:
While the default encoding as provided by this class performs the comparison on a prefix of the `byte[]`, alternate encoding schemes, may be implemented by a sub class which could involve possible discontinuous segments of the encoding. When such encoding is used partial matching may require a full, 16 byte encoding to be presented as the argument to this method.

Parameters:

`encoding` - a `byte[]` containing a possible encoding of an AID.

`offset` - the offset in the specified `byte[]` where the actual `AID` encoding starts in the encoding

Returns:

true iff the RID part of this AID matches the specified encoding.

Appendix B

Class AID.RID

```
java.lang.Object
  |
  +--com.sun.smartcard.util.Bytes
        |
        +--com.sun.smartcard.util.Buffer
              |
              +--com.sun.smartcard.util.ReadableBuffer
                    |
                    +--com.sun.smartcard.AID.RID
```

All Implemented Interfaces:

Readable

Enclosing class:

AID

---

public static class **AID.RID**

extends ReadableBuffer

Represent the RID (Registered ID) part of an AID.

This class is a wrapper around a serialized data representation, which may be shared with the parent AID.

A generic wrapper class for `byte[]`, `ReadableBuffer` is used as a base class.

Nested Class Summary

Nested classes inherited from class com.sun.smartcard.util.Readable
Readable.WithStream

Field Summary

Fields inherited from class com.sun.smartcard.util.Bytes

Constructor Summary

Private	**AID.RID**() Construct an empty instance, for exclusive use by the AID class.
	AID.RID(byte[] encoding) Construct an instance with a given data representation.

Method Summary

protected byte[]	**getData**() Share the data with the enclosing class.
short	**getLength**() Get the length of the data in the byte array for a RID (5).

Methods inherited from class com.sun.smartcard.util.ReadableBuffer
read

Methods inherited from class com.sun.smartcard.util.Buffer
getByteAt, getBytes, getDataSize, getFullyAt, getPrefixSkip, getSize, isValidIndex, match, match, match, setByteAt, setData

Methods inherited from class com.sun.smartcard.util.Bytes
equals, getBytes, getBytes, setBytes

Methods inherited from class java.lang.Object
clone, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

AID.RID

public **AID.RID**(byte[] encoding)

Construct an instance with a given data representation.

AID.RID

`private AID.RID()`

Construct an empty instance, for exclusive use by the AID class.

Method Detail

getLength

`public final short getLength()`

Get the length of the data in the byte array for a RID (5). The size of the RID is defined in ISO/IEC 7816-5

Overrides:

`getLength` in class `Bytes`

Returns:

5

---

getData

`protected final byte[] getData()`

Share the data with the enclosing class.

Overrides:

`getData` in class `Bytes`

Appendix C

Class AID.OwnerId

```
java.lang.Object
  |
  +--com.sun.smartcard.util.Bytes
        |
        +--com.sun.smartcard.util.Buffer
              |
              +--com.sun.smartcard.util.ReadableBuffer
                    |
                    +--com.sun.smartcard.AID.OwnerId
```

All Implemented Interfaces:

Readable

Enclosing class:

AID

---

public static class **AID.OwnerId**

extends ReadableBuffer

Represent the owner of an `Applet` instance that is contained in an `AID`. The owner of data in a Java Card Memory specifies the firewall that controls access to data.

This class is a wrapper around a serialized data representation, which may be shared with the parent AID. A generic wrapper class for `byte[]`, `ReadableBuffer` is used as a base class.

Nested Class Summary

Nested classes inherited from class com.sun.smartcard.util.Readable
`Readable.WithStream`

Field Summary

Fields inherited from class com.sun.smartcard.util.Bytes

Constructor Summary

private	**AID.OwnerId**() Construct an empty instance, for exclusive use by the AID class as wrapper on a shared data representation.
	AID.OwnerId(byte[] code) Construct an instance as wrapper on the specified data representation.

Method Summary

short	**getLength**() Get the length of the data in the byte array for the Owner part of the **AID**.
(package private) AID.RID	**getRID**() Get the RID part from the OwnerID.
boolean	**isIssuer**() Test for Card Issuer reserved value encoding.
boolean	**isNotControled**() Test for "not owner controlled" reserved value encoding.

Methods inherited from class com.sun.smartcard.util.ReadableBuffer
Read

Methods inherited from class com.sun.smartcard.util.Buffer
getByteAt, getBytes, getDataSize, getFullyAt, getPrefixSkip, getSize, isValidIndex, match, match, match, setByteAt, setData

Methods inherited from class com.sun.smartcard.util.Bytes
equals, getBytes, getBytes, getData, setBytes

Methods inherited from class java.lang.Object
clone, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

AID.OwnerId

`public` **`AID.OwnerId`**`(byte[] code)`

Construct an instance as wrapper on the specified data representation. This constructor is implemented by the constructor of the `ReadableBuffer` with the same signature.

AID.OwnerId

`private` **`AID.OwnerId`**`()`

Construct an empty instance, for exclusive use by the `AID` class as wrapper on a shared data representation.

Method Detail

getLength

`public short` **`getLength`**`()`

Get the length of the data in the byte array for the Owner part of the **AID**. This method implements the default encoding rules, with the OwnerId being a one byte extension of the RID and therefore returns the value six.

Overrides:

`getLength` in class `Bytes`

Returns:

6

---

getRID

```
final AID.RID getRID()
```

Get the RID part from the OwnerID.

---

isIssuer

```
public final boolean isIssuer()
```

Test for Card Issuer reserved value encoding.

---

isNotControled

```
public final boolean isNotControled()
```

Test for "not owner controlled" reserved value encoding.

Appendix D

Class AID.AppletId

```
java.lang.Object
  |
  +--com.sun.smartcard.util.Bytes
        |
        +--com.sun.smartcard.util.Buffer
              |
              +--com.sun.smartcard.util.ReadableBuffer
                    |
                    +--com.sun.smartcard.AID.AppletId
```

All Implemented Interfaces:

Readable

Enclosing class:

AID

---

public static class **AID.AppletId**

extends ReadableBuffer

Represent the unique reference to an `Applet` that is contained in an `AID`. Guaranteeing the uniqueness of an applet ID is left to the owner of the applet. The default implementation uses one additional byte appended to the encoding of the Owner.

This class is a wrapper around a serialized data representation, which may be shared with the parent AID. A generic wrapper class for `byte[]`, `ReadableBuffer` is used as a base class.

Nested Class Summary

Nested classes inherited from class com.sun.smartcard.util.Readable
Readable.WithStream

Field Summary

Fields inherited from class com.sun.smartcard.util.Bytes

Constructor Summary

private	**AID.AppletId**() Construct an empty instance, for exclusive use by the AID class.
	AID.AppletId(byte[] code) Construct an instance with a given data representation.

Method Summary

short	**getLength**() Get the length of the data in the byte array for the AppletId component of the AID.
(package private) AID.OwnerId	**getOwnerId**() Get the Owner Identifier contained in the applet identifier.
boolean	**isInformation**() Test for "not owner controlled" reserved value encoding.
boolean	**isManager**() Test for Card Issuer reserved value encoding.

Methods inherited from class com.sun.smartcard.util.ReadableBuffer
read

Methods inherited from class com.sun.smartcard.util.Buffer
getByteAt, getBytes, getDataSize, getFullyAt, getPrefixSkip, getSize, isValidIndex, match, match, match, setByteAt, setData

Methods inherited from class com.sun.smartcard.util.Bytes
equals, getBytes, getBytes, getData, setBytes

Methods inherited from class java.lang.Object
clone, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

AID.AppletId

`public` **`AID.AppletId`**`(byte[] code)`

Construct an instance with a given data representation.

AID.AppletId

`private` **`AID.AppletId`**`()`

Construct an empty instance, for exclusive use by the AID class.

Method Detail

getLength

`public short` **`getLength`**`()`

Get the length of the data in the byte array for the `AppletId` component of the `AID`. This method implements the default encoding rules, with the AppletId being a one byte extension of the OwnerID and therefore returns a value of seven.

Overrides:

`getLength` in class `Bytes`

Returns:

7

---

getOwnerId

`final AID.OwnerId` **`getOwnerId`**`()`

Get the Owner Identifier contained in the applet identifier.

---

isManager

```
public final boolean isManager()
```

Test for Card Issuer reserved value encoding.

---

isInformation

```
public final boolean isInformation()
```

Test for "not owner controlled" reserved value encoding.

Appendix E

Class AID.ConfigurationData

```
java.lang.Object
  |
  +--com.sun.smartcard.util.Bytes
        |
        +--com.sun.smartcard.util.Buffer
              |
              +--com.sun.smartcard.util.ReadableBuffer
                    |
                    +--com.sun.smartcard.AID.ConfigurationData
```

All Implemented Interfaces:

Readable

Direct Known Subclasses:

CardHolderDataApplet.CardHolderDataConfiguration

Enclosing class:

AID

---

public abstract static class **AID.ConfigurationData**

extends ReadableBuffer

Represent the `Applet` instance configuration data that may be contained in an `AID`. The configuration data, when present in an AID may be used when the Applet is instantiated to configure its components. For instance, the size of an array of object that represent a set of user data may be specified in the configuration. This class to be extended when a configuration is present on the encoding of an **AID**.

This class is a wrapper around a serialized data representation, which may be shared with the parent AID. A generic wrapper class for `byte[]`, `ReadableBuffer` is used as a base class;

The current version of the implementation does not support configuration in the default encoding, however in a more mature version this might well be different.

Nested Class Summary

Nested classes inherited from class com.sun.smartcard.util.Readable
Readable.WithStream

Field Summary

Fields inherited from class com.sun.smartcard.util.Bytes

Constructor Summary

protected	**AID.ConfigurationData**() Construct a default instance.
	AID.ConfigurationData(byte[] code) Construct an instance as a wrapper around a given data representation

Method Summary

short	**getLength**() Specify the length of the data (byte[]) that represents the configuration data.
short	**getPrefixSkip**() Skip the initial portion of the AID encoding

Methods inherited from class com.sun.smartcard.util.ReadableBuffer
read

Methods inherited from class com.sun.smartcard.util.Buffer
getByteAt, getBytes, getDataSize, getFullyAt, getSize, isValidIndex, match, match, match, setByteAt, setData

Methods inherited from class com.sun.smartcard.util.Bytes

equals, getBytes, getBytes, getData, setBytes

Methods inherited from class java.lang.Object
clone, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

AID.ConfigurationData

public **AID.ConfigurationData**(byte[] code)

Construct an instance as a wrapper around a given data representation. This constructor is implemented by the constructor of the ReadableBuffer with the same signature. The data specified by the argument is not copied.

AID.ConfigurationData

protected **AID.ConfigurationData**()

Construct a default instance.

Method Detail

getLength

public short **getLength**()

Specify the length of the data (byte[]) that represents the configuration data. The default encoding does not contain any configuration data, so its length is 0.

Overrides:

getLength in class Bytes

---

getPrefixSkip

public short **getPrefixSkip**()

Skip the initial portion of the AID encoding. The value returned is in accordance with the default encoding and is equal to the length of the AppletID representation plus one byte for the function code.

Overrides:

`getPrefixSkip` in class `Buffer`

Returns:

the offset in the `AID` data representation where the configuration data can be found.

Appendix F

Class AIDInterpreter

```
java.lang.Object
   |
   +--com.sun.smartcard.AIDInterpreter
```

All Implemented Interfaces:

java.lang.Cloneable

---

public class **AIDInterpreter**

extends java.lang.Object

implements java.lang.Cloneable

Provide access to the data describing an Applet as specified by the AID. This class is intended for use in terminals, to provide detailed information about an applet in a card. Applet information is partially encoded in the AID data string read from the card; additional data regarding the Applet may be embedded in a sub class derived from this class or in part retrieved on the internet for instance using references embedded in the sub class implementation. The attributes defined for this class provide explicit support for internationalization of the terminal code.

Together with the in-card companion class `com.sun.smarcard.AID` this class is a key component in providing generic application management support for the Java Card platform. The class provides support for the dynamic configuration of card terminals.

The definition of sub classes of this class, their class files, may be stored at web-accessible repositories. A class naming scheme is implemented in this class that is based on the internationally registered identifier (RID) that is included in each AID encoding. A factory method is provided to instantiate a class based on a specific RID using this naming scheme. The class definition register may be maintained by an international organization like Java Card Forum.

By default the data describing the applet and its in-terminal components is made available by this class as a Java attribute list. Each attribute consist of an attribute name and attribute value both attribute parts being a Java String. Sub classes may define additional, e.g method call based, access to part of the data.

The following types of properties are specified here:

1. Owner properties
2. Applet Properties
3. Terminal properties

Card application attributes

The following attribute names are defined by this class. This class actually supports a basic set of attributes; a sub-class may implement support for additional attributes. Attributes that are not supported have a value of `null`.

Owner

The name of the organizational or business unit that is responsible for the Applet.

Owner.code

The hexadecimal representation of the encoding of the ownerID.

Owner.URL

A URL referring to a description of the organizational or business unit that is responsible for the Applet.

Owner.Logo

A URL referring to an image characterizing the organizational or business unit that is responsible for the Applet.

Owner.BackOfficeURI

A URI referring to the back office (application management system) operated by the Applet owner to manage the use of the Applet in the card.

Applet.Function.DescriptionURL

A web page describing the function of the applet in a default langauge.

Applet.Function

A brief description of the Applet function in a default langauge.

Applet.Function.+*lang*+

A brief description of the Applet function in the language specified by +*lang*+ (encoded as ISO language identifiers).

Applet.CAPFile.URL

A URL specifying the name of the file containing the downloadable code for the applet.

Applet.Title

A summary description of the Applet.

Applet.Title.+*lang*+

A summary description of the Applet in the language specified by +*lang*+.

Applet.Proxy.Count

The number of application proxies that may be used to interact with the applet. Each proxy may be offering a different set of functions.

Applet.Proxy.Title.+count+

A summary description of the application proxy.

Applet.Proxy.Title.+count+.+*lang*+

A summary description of the application proxy in the language specified by +*lang*+.

Applet.Proxy.CardApplicationServiceAPI.+count+

The class name of the Java interface that specifies the functional interface of the specific application proxy.

Applet.Proxy.Class.+count+

The class name of a specific application proxy.

Applet.Proxy.Jar.+count+

The URL of a jar file that contains the proxy java class and supporting classes for a specific application proxy.

Applet.Function.Code

The hex representation of the one byte Applet function code as defined in ISO/IEC 14344.

Applet.Function.ProxyClass

A Java class name of the specific class that can be used to communicate optimally with the applet and its backoffice. This class in general wil be provided by the Application Provider; instantiating the class in a terminal may be subject to commercial agreements.

Applet.Function.ProxyClass.GenericInterface

A Java class name of the specific interface that may be used to communicate with the applet to obtain the shared functionality as defined by the Function code.

Applet.Function.ProxyClass.GenericInterface.Implementation

A Java class name of an implementation of the generic interface that may be used to communicate with the applet to obtain the shared functionality as defined by the Function code. The instantiation of this class in the terminal should be possible without commercial restrictions.

Applet.Version

The decimal representation of the version in a "major.minor" notation. This is 0.0 if not supported by the AID.

RID

The name of the legal entity that has obtained the RID.

RID.code

The hexadecimal representation of the RID

RID.URL

A URL referring to a description of the legal entity that has obtained the RID.

PIX

The hexadecimal representation of the PIX.

Field Summary	
private AID	**aid** The AID being interpreted.
Static java.lang.String	**AID_CLASS_NAME_PREFIX**
Static java.lang.String[]	**AID_INTERPRETER_PACKAGE_PATH** The name of a JavaCard special package for Issuer specific versions of this class.

Constructor Summary	
AIDInterpreter()	

Method Summary	
protected static AID	**getAID**(AID.RID rid) Get an instance of an AID sub class as appropriate for the applet.
protected static java.lang.String[]	**getAIDPackageList**() Get a list of all the package names that may contain the definition of an appropriate AIDInterpreter class definition.
java.lang.String[]	**getAppletStateDataPropertyNames**() Obtain the attribute names for applet data that may be included in the AID.
java.lang.String	**GetAppletString**(short stringreference) Obtain a string that has been removed from the applet code by the conversion to Java Card ByteCode.
java.util.Properties	**GetAsProperties**() Interpret the content of an AID as a list of name value pairs.
AIDInterpreter	**GetCardEncodingInterpreter**(byte[] encoding) Clone the interpreter initialized with the encoding of a compatible AID as may be obtained from the card.
java.lang.String[]	**GetConfigurationPropertyNames**() Obtain the list of attribute names that describe the values that may be present in the configuration part of the AID.
(package private) static AIDInterpreter	**makeOne**(AID.RID rid) Create an instance of a derived class for the AID Interpreter, as defined by the specified RID.
static AIDInterpreter	**makeOne**(byte[] encoding) Create an instance of the AID that fits a specified AID encoding.

Methods inherited from class java.lang.Object
Clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

AID_INTERPRETER_PACKAGE_PATH

public static final java.lang.String[] **AID_INTERPRETER_PACKAGE_PATH**

The name of a JavaCard special package for Issuer specific versions of this class.

AID_CLASS_NAME_PREFIX

```
public static final java.lang.String AID_CLASS_NAME_PREFIX
```

aid

```
private AID aid
```

The AID being interpreted.

Constructor Detail

AIDInterpreter

```
public AIDInterpreter()
```

Method Detail

makeOne

```
static final AIDInterpreter makeOne(AID.RID rid)
```

Create an instance of a derived class for the AID Interpreter, as defined by the specified RID.

The interpreter classes are located in a Java package with the special name "javacard.aidinterpreters".

Note:
This method is never called inside the card.

---

getAIDPackageList

```
protected static java.lang.String[] getAIDPackageList()
```

Get a list of all the package names that may contain the definition of an appropriate `AIDInterpreter` class definition.

---

makeOne

```
public static final AIDInterpreter makeOne(byte[] encoding)
```

Create an instance of the AID that fits a specified AID encoding.

Note:
This method is never called inside the card. The in-card implementation of this class may dispense with it.

---

getAID

```
protected static AID getAID(AID.RID rid)
```

Get an instance of an AID sub class as appropriate for the applet. This class will be defined by the application provider as part of the applet code. Instances of this specific class will both be used in the card and outside it, as is demonstrated by the interpreter class.

---

getCardEncodingInterpreter

```
public AIDInterpreter getCardEncodingInterpreter(byte[] encoding)
```

Clone the interpreter initialized with the encoding of a compatible AID as may be obtained from the card. A compatible AID encoding matches the Applet ID.

Returns:

null if the specified encoding does not match the application ID.

---

getAsProperties

```
public java.util.Properties getAsProperties()
```

Interpret the content of an AID as a list of name value pairs. This method returns a properties object initialized with the properties available by interpreting the default AID encoding. Additional properties may be provided in a sub class, e.g by adding properties to the result of this function in the constructor.

getConfigurationPropertyNames

```
public java.lang.String[] getConfigurationPropertyNames()
```

Obtain the list of attribute names that describe the values that may be present in the configuration part of the AID. This method is for out of card use, providing interoperable interpretation of the result in terminals. This method uses the default encoding.

getAppletStateDataPropertyNames

```
public java.lang.String[] getAppletStateDataPropertyNames()
```

Obtain the attribute names for applet data that may be included in the AID. By default this method does nothing and returns `null`. A sub class that is intended to interpret an AID that contains optional state data in the encoding of the AID should override this method.

getAppletString

```
public java.lang.String getAppletString(short stringreference)
```

Obtain a string that has been removed from the applet code by the conversion to Java Card ByteCode. This method allows the limited support of the Java `String` class in Java Card applet code whereby the strings embedded in the code are replaced by a numeric reference. The numeric reference may be returned to the terminal in a response and subsequently used as index into an array of strings that may be made available to the card terminal.